Capstone project report

Introduction

Once every few days, Starbucks sends out an offer to users of the mobile app. An offer can be merely an advertisement for a drink or an actual offer such as a discount or BOGO (buy one get one free) or just an informational offer which includes the product information. Some users might not receive any offers during certain weeks. In this way, Starbucks can probably increase the possibility that the customer opens the offer after they receive it and eventually finish the transaction. It will also help improve customer loyalty by keep reminding them of the latest product information. But the point here is how to send out the offer in a smarter way, which means, how to maximize the possibility that customer opens the offer and finish the transactions. Therefore, we'll try to analyze the Starbucks history dataset to see if we could get some insight from it.

Business Context

- The program used to create the data simulates how people make purchasing decisions and how those decisions are influenced by promotional offers.
- Each person in the simulation has some hidden traits that influence their purchasing patterns and are associated with their observable traits. People produce various events, including receiving offers, opening offers, and making purchases.
- As a simplification, there are no explicit products to track. Only the amounts of each transaction or offer are recorded.
- There are three types of offers that can be sent: buy-one-get-one (BOGO), discount, and informational. In a BOGO offer, a user needs to spend a certain amount to get a reward equal to that threshold amount. In a discount, a user gains a reward equal to a fraction of the amount spent. In an informational offer, there is no reward, but neither is there a required amount that the user is expected to spend. Offers can be delivered via multiple channels.

Project Goal

Based on the context above, this project will try to ask the questions below

- What factors mainly affect the usage of the offer from the customer? Should the company send out the offer or not?
- How possible will a customer open and use the offer sent to them? Are there any common characteristics of the customers who take the offer?

Data Dictionary

The data is contained in three files:

- portfolio.json—containing offer ids and meta data about each offer (duration, type, etc.)
- profile.json—demographic data for each customer
- transcript.json—records for transactions, offers received, offers viewed, and offers completed

Here is the schema and explanation of each variable in the files:

portfolio.json

- id (string)—offer id
- offer_type (string)—type of offer ie BOGO, discount, informational
- difficulty (int)—minimum required spend to complete an offer
- reward (int)—reward given for completing an offer
- duration (int)—time for offer to be open, in days
- channels (list of strings)

profile.json

- age (int)—age of the customer
- became_member_on (int)—date when customer created an app account
- gender (str)—gender of the customer (note some entries contain 'O' for other rather than M or F)
- id (str)—customer id
- income (float)—customer's income

transcript.json

- event (str)—record description (ie transaction, offer received, offer viewed, etc.)
- person (str)—customer id
- time (int)—time in hours since start of test. The data begins at time t=0
- value—(dict of strings)—either an offer id or transaction amount depending on the record

Data Exploration

In order to analyze the problem better in next sections, first need to explore the datasets which includes checking the missing value, visualizing the data distribution, etc. In that way, we can have a better understanding on how the dataset looks like and how to select the important features to support the model implementation.

[3]:
```
# view the portfolio dataset
portfolio.head()
```

[3]:

	channels	difficulty	duration	id	offer_type	reward
0	[email, mobile, social]	10	7	ae264e3637204a6fb9bb56bc8210ddfd	bogo	10
1	[web, email, mobile, social]	10	5	4d5c57ea9a6940dd891ad53e9dbe8da0	bogo	10
2	[web, email, mobile]	0	4	3f207df678b143eea3cee63160fa8bed	informational	0
3	[web, email, mobile]	5	7	9b98b8c7a33c4b65b9aebfe6a799e6d9	bogo	5
4	[web, email]	20	10	0b1e1539f2cc45b7b9fa7c272da2e1d7	discount	5

[4]:
```
# quick check on missing value in dataset
portfolio.isnull().sum()
```

[4]:
```
channels      0
difficulty    0
duration      0
id            0
offer_type    0
reward        0
dtype: int64
```

As shown above, there are no missing values in the portfolio dataset.

8]:
```
# view the profile dataset which contains demographic information
profile.head()
```

8]:

	age	became_member_on	gender	id	income
0	118	20170212	None	68be06ca386d4c31939f3a4f0e3dd783	NaN
1	55	20170715	F	0610b486422d4921ae7d2bf64640c50b	112000.0
2	118	20180712	None	38fe809add3b4fcf9315a9694bb96ff5	NaN
3	75	20170509	F	78afa995795e4d85b5d9ceeca43f5fef	100000.0
4	118	20170804	None	a03223e636434f42ac4c3df47e8bac43	NaN

By viewing the first several rows of the dataset, it apparently shows missing values in the age column which is encoded as 118, and there are missing values in income column too.

[11]:
```
# number of missing values in age column
profile[profile.age==118].count()
```

t[11]:
```
age                 2175
became_member_on    2175
gender                 0
id                  2175
income                 0
dtype: int64
```

[12]:
```
# check if the rows which have missing age also have missing gender and income
profile[profile.age==118].head()
```

t[12]:

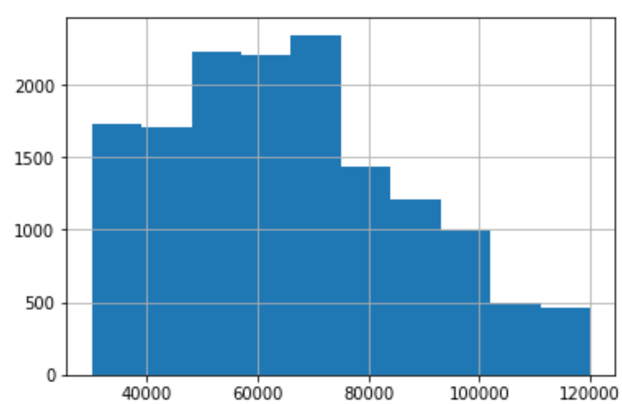
	age	became_member_on	gender	id	income
0	118	20170212	None	68be06ca386d4c31939f3a4f0e3dd783	NaN
2	118	20180712	None	38fe809add3b4fcf9315a9694bb96ff5	NaN
4	118	20170804	None	a03223e636434f42ac4c3df47e8bac43	NaN
6	118	20170925	None	8ec6ce2a7e7949b1bf142def7d0e0586	NaN
7	118	20171002	None	68617ca6246f4fbc85e91a2a49552598	NaN

Apparently, the rows which have missing age also missing gender and income, which means probably it's fine to just drop the rows in the following steps to support the model implementation.

Get a quick check on how the income distribution looks like in the dataset.

In [13]:
```
# get a quick view on income distribution
profile.income.hist()
```

Out[13]: `<matplotlib.axes._subplots.AxesSubplot at 0x7f7b0c0512b0>`



Then take a quick view on the transcript dataset.

[14]:
```
# view the transcript dataset
transcript.head()
```

[14]:

	event	person	time	value
0	offer received	78afa995795e4d85b5d9ceeca43f5fef	0	{'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}
1	offer received	a03223e636434f42ac4c3df47e8bac43	0	{'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}
2	offer received	e2127556f4f64592b11af22de27a7932	0	{'offer id': '2906b810c7d4411798c6938adc9daaa5'}
3	offer received	8ec6ce2a7e7949b1bf142def7d0e0586	0	{'offer id': 'fafdcd668e3743c1bb461111dcafc2a4'}
4	offer received	68617ca6246f4fbc85e91a2a49552598	0	{'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}

Since the value columns include multiple information which should be extracted out for clearer and easier analysis, first do some basic manipulation on the dataset.

.7]:
```
# extract the different values in value column out
transcript = pd.concat([transcript, transcript['value'].apply(pd.Series)], axis=1)
transcript.head()
```

.7]:

	event	person	time	value	offer id	amount	offer_id	reward
0	offer received	78afa995795e4d85b5d9ceeca43f5fef	0	{'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}	9b98b8c7a33c4b65b9aebfe6a799e6d9	NaN	NaN	NaN
1	offer received	a03223e636434f42ac4c3df47e8bac43	0	{'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}	0b1e1539f2cc45b7b9fa7c272da2e1d7	NaN	NaN	NaN
2	offer received	e2127556f4f64592b11af22de27a7932	0	{'offer id': '2906b810c7d4411798c6938adc9daaa5'}	2906b810c7d4411798c6938adc9daaa5	NaN	NaN	NaN
3	offer received	8ec6ce2a7e7949b1bf142def7d0e0586	0	{'offer id': 'fafdcd668e3743c1bb461111dcafc2a4'}	fafdcd668e3743c1bb461111dcafc2a4	NaN	NaN	NaN
4	offer received	68617ca6246f4fbc85e91a2a49552598	0	{'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}	4d5c57ea9a6940dd891ad53e9dbe8da0	NaN	NaN	NaN

Data Preprocessing

In order to find out what mainly affect the finish of the transaction by sending the offer, in the data processing process, also need to process the data to merge the events of each specific offer sent so as to find out which offer was received, viewed and finally completed with a transaction.

21]:
```
# quick view on event distribution by offer_type by combining two dataset (portfolio, transcript)
transcript=transcript.merge(portfolio,how='left',left_on='offer_id',right_on='id')
transcript.groupby(['event','offer_type'])['offer_type'].count()
```

21]:
```
event            offer_type
offer completed  bogo             15669
                 discount         17910
offer received   bogo             30499
                 discount         30543
                 informational    15235
offer viewed     bogo             25449
                 discount         21445
                 informational    10831
Name: offer_type, dtype: int64
```

Since offer_id is not associated with any 'transaction' event, in order to flag whether the offer has been finally completed with a transaction, here we need to link the offer id back to all transaction events. For BOGO and discount offer, both of them will have the consequence of offers received, viewed, transaction and offer completed which will apparently show that the offer is redeemed and should definitely be sent out. For the information offer, though there's no reward step there should still be a transaction that is linked to the usage of the offer.

27]:
```python
transcript_processed = transcript_processed.merge(portfolio, how = 'left', left_on='offer_id', right_on='id')
transcript_processed['duration'] = np.where(transcript_processed['duration_x'].isnull(), \
                                            transcript_processed['duration_y'], transcript_processed['duration_x'])
transcript_processed.drop(columns=['duration_x','offer_type_x','difficulty_x','channels_x','duration_y'],\
                          axis=1, inplace=True)
transcript_processed.rename(columns={'channels_y':'channels','reward_y':'reward','difficulty_y':'difficulty','offer_ty
```

28]:
```python
# quick check on processed dataset
transcript_processed.head()
```

28]:

	event	person	time	value	amount	id_x	
0	offer received	0009655768c64bdeb2e877511632db8f	168	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}	NaN	5a8bc65990b245e5a138643cd4eb9837	5a8bc65990b245e5a
1	offer viewed	0009655768c64bdeb2e877511632db8f	192	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}	NaN	5a8bc65990b245e5a138643cd4eb9837	5a8bc65990b245e5a
2	transaction	0009655768c64bdeb2e877511632db8f	228	{'amount': 22.16}	22.16	NaN	5a8bc65990b245e5a
3	offer received	0009655768c64bdeb2e877511632db8f	336	{'offer id': '3f207df678b143eea3cee63160fa8bed'}	NaN	3f207df678b143eea3cee63160fa8bed	3f207df678b143ee
4	offer viewed	0009655768c64bdeb2e877511632db8f	372	{'offer id': '3f207df678b143eea3cee63160fa8bed'}	NaN	3f207df678b143eea3cee63160fa8bed	3f207df678b143ee

Next, after we get the data together, we need to extract the transactions which were completed after the offer was received and viewed. Since we've already filled all transaction's offer id, we can extract the transactions converted from offers by checking if the offer id before the transaction is the same as the transaction's offer id.

```
[29]: # subset the dataset with only offer viewed, transaction, and offer completed events
transactions_after_viewed = transcript_processed[(transcript_processed['event']=='offer viewed')|\
                                                 (transcript_processed['event']=='transaction')|\
                                                 (transcript_processed['event']=='offer completed')].copy()

# generate the previous offer id
transactions_after_viewed['pre_offer_id'] = transactions_after_viewed.groupby(['person', 'offer_id'])['offer_id'].shift

# create flag for responsed offer which competed after customer viewing the offer
transactions_after_viewed['completed_offer'] = np.where(transactions_after_viewed['pre_offer_id']==\
                                                       transactions_after_viewed['offer_id'],1,0)
```

```
[31]: # join back the 'offer received' events which was filtered out in the previous step
offer_received = transcript_processed[transcript_processed['event']=='offer received']

offer_received['pre_offer_id']=np.nan
offer_received['completed_offer']=np.nan

transcript_processed = offer_received.append(transactions_after_viewed).sort_values(['person','time'])
transcript_processed.head()
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  after removing the cwd from sys.path.
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  """
```

[31]:

	event	person	time	value	amount	id_x	
0	offer received	0009655768c64bdeb2e877511632db8f	168	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}	NaN	5a8bc65990b245e5a138643cd4eb9837	5a8bc65990b245e5a
1	offer viewed	0009655768c64bdeb2e877511632db8f	192	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}	NaN	5a8bc65990b245e5a138643cd4eb9837	5a8bc65990b245e5a
2	transaction	0009655768c64bdeb2e877511632db8f	228	{'amount': 22.16}	22.16	NaN	5a8bc65990b245e5a
3	offer received	0009655768c64bdeb2e877511632db8f	336	{'offer id': '3f207df678b143eea3cee63160fa8bed'}	NaN	3f207df678b143eea3cee63160fa8bed	3f207df678b143ee
4	offer	0009655768c64bdeb2e877511632db8f	372	{'offer id':	NaN	3f207df678b143eea3cee63160fa8bed	3f207df678b143ee

Since the different offer has difference consequence of completion, for example, for the informational offer, there'll not be rewards. Therefore, separate the transcript data by offer type for easier analysis.

```
32]: #split transcript into 3 different offer types
bogo = transcript_processed[transcript_processed['offer_type']=='bogo'].copy()
discount = transcript_processed[transcript_processed['offer_type']=='discount'].copy()
informational = transcript_processed[transcript_processed['offer_type']=='informational'].copy()
```

Within each offer type, use responded_offer flagged in previous steps we can filter out the offers which were successfully viewed and completed by users. For BOGO and discount offer, the responsed offer should be the one that with 'offer complete' events, and for the informational offer, just 'transaction' can be seen as a successful offer.

5]:
```
# extract responded offer under bogo and informational type
bogo_completed = bogo[['person','offer_id']][(bogo['completed_offer']==1) & (bogo['event']=='offer completed')].groupby
discount_completed = discount[['person','offer_id']][(discount['completed_offer']==1) & (discount['event']=='offer comp
```

Next, will separate out customers who only viewed the offers without transaction and completion at the end and the customers who only received the offer without viewing it.

37]:
```
# filter out offer with transactions or completed, and offer which have viewed events
bogo_ids_transaction_completed = bogo[['person','offer_id']][(bogo['event']=='transaction') | \
                                 (bogo['event']=='offer completed') ].groupby(['person','offer_id']).count().reset_index()
bogo_ids_received = bogo[['person','offer_id']][bogo['event']=='offer received'].groupby(['person','offer_id']).count(

# get the offer records which was only viewed without transaction and completion
bogo_merged = bogo_ids_transaction_completed.merge(bogo_ids_received,how='right',on=['person','offer_id'],indicator=Tr
bogo_merged.head()
```

37]:

	person	offer_id	_merge
0	0009655768c64bdeb2e877511632db8f	f19421c1d4aa40978ebb69ca19b0e20d	both
1	00116118485d4dfda04fdbaba9a87b5c	f19421c1d4aa40978ebb69ca19b0e20d	both
2	0011e0d4e6b944f998e987f904e8c1e5	9b98b8c7a33c4b65b9aebfe6a799e6d9	both
3	0020c2b971eb4e9188eac86d93036a77	4d5c57ea9a6940dd891ad53e9dbe8da0	both
4	0020ccbbb6d84e358d3414a3ff76cffd	9b98b8c7a33c4b65b9aebfe6a799e6d9	both

Then, based on merged dataset above, we can separate out customers who only viewed the offer after they received the offer and customers who didn't even open the offer after they receive the offer.

[39]:
```
# subtract the customers who received the offer without finishing the transaction
bogo_no_transaction = bogo_merged[bogo_merged['_merge']=='right_only']
bogo_no_transaction = bogo_no_transaction.merge(transcript_processed, how='left',on=['person','offer_id'])

# subset users who only veiwed the offer and people who viewed the offer but didn't make transaction
bogo_viewed = bogo_no_transaction[['person','offer_id']][bogo_no_transaction['event']=='offer viewed']\
               .groupby(['person','offer_id']).count().reset_index()
bogo_no_transaction.drop(['_merge'], axis=1, inplace=True)

# get the customers who didn't even view the offer after they received the BOGO offer
bogo_no_view = bogo_no_transaction.merge(bogo_viewed[['person','offer_id']], how='left', indicator=True)
bogo_no_view = bogo_no_view[bogo_no_view['_merge']=='left_only'].copy()
```

For these steps, will do the same manipulation for both BOGO and discount offer. After above processing, filter out the transaction regardless of receiving or viewing the offer.

1]:
```
# subset the offer which has no correction with offer
bogo_unrelated = bogo[['person','offer_id']][(bogo['event']=='offer completed') & (bogo['completed_offer']!=1)]\
                  .groupby(['person','offer_id']).count().reset_index()
discount_unrelated = discount[['person','offer_id']][(discount['event']=='offer completed')\
                  & (discount['completed_offer']!=1)].groupby(['person','offer_id']).count().reset_index()
```

After separating the different cases of customers, the following steps will firstly focus on customers who finish the transaction after receiving the offer and customers who only view the offer without any transaction.

```
In [42]: # combine the two kind of customers cases which are focused on
bogo_completed['offer_responded']=1
bogo_viewed['offer_responded']=0
bogo_offer = bogo_completed.append(bogo_viewed, sort=False)

discount_completed['offer_responded']=1
discount_viewed['offer_responded']=0
discount_offer = discount_completed.append(discount_viewed, sort=False)
```

As for the informational offer, the offer could only be counted as responded under the effect of the offer when the transaction is finished within the duration of the offer.

Feature engineering

After basic processing, the next step will look if there are any columns that can be used to create new features.

- generate a new column for the length of customer's membership

```
[51]: bogo_offer.head()
```

[51]:

	person	offer_id	offer_responded	age	gender	income	membership_days
0	0011e0d4e6b944f998e987f904e8c1e5	9b98b8c7a33c4b65b9aebfe6a799e6d9	1	40	O	57000.0	39.0
1	0020c2b971eb4e9188eac86d93036a77	4d5c57ea9a6940dd891ad53e9dbe8da0	1	59	F	90000.0	824.0
2	0020ccbbb6d84e358d3414a3ff76cffd	9b98b8c7a33c4b65b9aebfe6a799e6d9	1	24	F	60000.0	771.0
3	0020ccbbb6d84e358d3414a3ff76cffd	f19421c1d4aa40978ebb69ca19b0e20d	1	24	F	60000.0	771.0
4	004b041fbfe44859945daa2c7f79ee64	f19421c1d4aa40978ebb69ca19b0e20d	1	55	F	74000.0	158.0

- generate the count of the offer received for each user

```
In [53]: # generate the count of offers received per person
offer_cnt=transcript_processed[transcript_processed['event']=='offer received'].groupby(['person','offer_id','time']).
offer_cnt.rename(columns={'event':'offer_received_cnt'},inplace=True)

offer_cnt.drop(['time'], axis=1, inplace=True)

# ensure only unique person-offer_id pairs
offer_cnt=offer_cnt.groupby(['person','offer_id']).sum().reset_index()
offer_cnt.head()
```

Out[53]:

	person	offer_id	offer_received_cnt
0	0009655768c64bdeb2e877511632db8f	2906b810c7d4411798c6938adc9daaa5	1
1	0009655768c64bdeb2e877511632db8f	3f207df678b143eea3cee63160fa8bed	1
2	0009655768c64bdeb2e877511632db8f	5a8bc65990b245e5a138643cd4eb9837	1
3	0009655768c64bdeb2e877511632db8f	f19421c1d4aa40978ebb69ca19b0e20d	1
4	0009655768c64bdeb2e877511632db8f	fafdcd668e3743c1bb461111dcafc2a4	1

- subtract the transactions which's not related to the offer

```
54]: # subtract the transactions which's not related to the offer
transactions_not_related=transcript_processed[(transcript_processed['event']=='transaction') & (transcript_processed[
transactions_not_related.rename(columns={'amount':'amount_invalid'},inplace=True)
```

- calculate the time lap between offers

```
[55]: # convert time into days
transcript_processed['day_offer']=transcript_processed['time']/24
# drop unnecessary columns
transcript_processed.drop(['time'], axis=1, inplace=True)

# calculate the time between offers
transcript_processed['time_gap']=transcript_processed[transcript_processed['event']=='offer received'].groupby(['

# fill missing values with 0
transcript_processed['time_gap']=transcript_processed['time_gap'].fillna(value=0)

df_time_gap=transcript_processed.groupby(['person','offer_id'])['time_gap'].sum().reset_index()
```

- Merge the temporary data created above together, then drop the missing values in gender column, and split the channel column to the categorical variable

56]:
```
# merge to get offers received count and invalid amount transacted
bogo_offer=bogo_offer.merge(offer_cnt[['person','offer_id','offer_received_cnt']],how='left',on=['person','offer_id'])
bogo_offer=bogo_offer.merge(transactions_not_related[['person','offer_id','amount_invalid']],how='left',on=['person','
```

57]:
```
# fill missing values for amount_invalid with 0
bogo_offer['amount_invalid']=bogo_offer['amount_invalid'].fillna(value=0)
bogo_offer.dropna(inplace=True)
```

58]:
```
bogo_offer.head()
```

58]:

	person	offer_id	offer_responded	age	gender	income	membership_days	offer_received_cnt	amount_invalid
	e0d4e6b944f998e987f904e8c1e5	9b98b8c7a33c4b65b9aebfe6a799e6d9	1	40	O	57000.0	39.0	1	0.0
	2b971eb4e9188eac86d93036a77	4d5c57ea9a6940dd891ad53e9dbe8da0	1	59	F	90000.0	824.0	1	0.0
	Dccbbb6d84e358d3414a3ff76cffd	9b98b8c7a33c4b65b9aebfe6a799e6d9	1	24	F	60000.0	771.0	1	0.0
	Dccbbb6d84e358d3414a3ff76cffd	f19421c1d4aa40978ebb69ca19b0e20d	1	24	F	60000.0	771.0	1	0.0
	)041fbfe44859945daa2c7f79ee64	f19421c1d4aa40978ebb69ca19b0e20d	1	55	F	74000.0	158.0	1	0.0

60]:
```
# merge with portfolio to get offer details
bogo_offer=bogo_offer.merge(portfolio,how='left',on='offer_id')

# convert channels into categorical variables
channels = bogo_offer['channels'].apply(pd.Series)
channels = channels.rename(columns={0:'web',1:'email',2:'mobile',3:'social'})
bogo_offer = pd.concat([bogo_offer[:], channels[:]], axis=1)
rename('web',bogo_offer)
rename('email',bogo_offer)
rename('mobile',bogo_offer)
rename('social',bogo_offer)
bogo_offer = bogo_offer.drop(['channels'], axis=1, inplace=False)

# convert gender into categorical variables
bogo_offer=dummy(bogo_offer,'gender')
```

61]:
```
# quick check on processed data
bogo_offer.head()
```

61]:

come	membership_days	offer_received_cnt	amount_invalid	difficulty	duration	offer_type	reward	web	email	mobile	social	gender_F	gender_M	gender_O
'000.0	39.0	1	0.0	5	7	bogo	5	1	1	1	0	0	0	1
)000.0	824.0	1	0.0	10	5	bogo	10	1	1	1	1	1	0	0
)000.0	771.0	1	0.0	5	7	bogo	5	1	1	1	0	1	0	0
)000.0	771.0	1	0.0	5	5	bogo	5	1	1	1	1	1	0	0
1000.0	158.0	1	0.0	5	5	bogo	5	1	1	1	1	1	0	0

Building model

After pre-processing the data, the next step we'll start to implement models to figure out which factors affect most whether the customer will respond to the offer or not. And this project also attempts to predict whether the customer will respond to the different types of offers or not.

Therefore, we'll use the 'offer_responded' flag in the dataset to build models to predict if the customer will respond to the offer of not. Here we will choose the basic tree model as a baseline which will help explain the feature importance better so that we can get some insight into what factors affect customer's behavior most.

Model implementation preparation

- Prepare the date set, set the features variable and target columns

```
[72]: def data_prep(df,drop_cols_prep):
    '''
    inputs:
    - df: prepared dataframe for modeling

    outputs:
    - Returns 2 dataframes - features and target dataframes
    '''
    # Split the data into features and target label
    target = df['offer_responded']
    features = df.drop(drop_cols_prep, axis=1, inplace=False)
    return features,target
```

- Split the data into training and test sets

```
[73]: def model_pipeline(features,target):
    '''
    inputs:
    - features & target dataframe

    outputs:
    - Splits features and target dataframe to train and test sets, performs feature scaling on both datasets.
    - Outputs X_train, X_test, y_train and y_test dataframes
    '''

    #split into training and test sets
    X_train, X_test, y_train, y_test = train_test_split(features,target, test_size=0.20, random_state=42)

    #fit and transform scaling on training data
    scaler=StandardScaler()
    X_train=scaler.fit_transform(X_train)

    #scale test data
    X_test=scaler.transform(X_test)
    return X_train,X_test,y_train, y_test
```

- Create a function to execute the model for different offer types

```python
[74]: # reference: Udacity -- 'Finding Donors for Charity ML' project
      # reference: Udacity -- 'Creating Customer Segments with Arvato' project
      def train_predict(model, X_train, y_train, X_test, y_test):
          '''
          inputs:
             - model: the model to be trained and predicted on
             - sample_size: the size of samples (number) to be drawn from training set
             - X_train: features training set
             - y_train: review_scores_rating training set
             - X_test: features testing set
             - y_test: review_scores_rating testing set
          '''
          results = {}

          #Fit the model to the training data and get training time
          start = time()
          model = model.fit(X_train, y_train)
          end = time()
          results['train_time'] = end-start

          # Get predictions on the test set(X_test), then get predictions on first 300 training samples
          start = time()
          predictions_test = model.predict(X_test)
          predictions_train = model.predict(X_train)
          end = time()

          # Calculate the total prediction time
          results['pred_time'] = end-start

          #add training accuracy to results
          results['training_score']=model.score(X_train,y_train)

          #add testing accuracy to results
          results['testing_score']=model.score(X_test,y_test)

          print("{} trained on {} samples.".format(model.__class__.__name__, len(y_train)))
          print("MSE_train: %.4f" % mean_squared_error(y_train,predictions_train))
          print("MSE_test: %.4f" % mean_squared_error(y_test,predictions_test))
          print("Training accuracy:%.4f" % results['training_score'])
          print("Test accuracy:%.4f" % results['testing_score'])
          print(classification_report(y_test, predictions_test,digits=4))
          return results
```

```python
[75]: def run_model(clf1,clf2,name):
          '''
          inputs:
          - clf1: first classifier model
          - clf2: 2nd classifier model for comparison
          - name: name of models for comparison

          outputs:
          - Dataframe of results from model training and prediction
          '''

          # Collect results from models
          results = {}
          for clf in [clf1, clf2]:
              clf_name = clf.__class__.__name__ + '_' +name
              results[clf_name] = {}
              results[clf_name]= train_predict(clf, X_train, y_train, X_test, y_test)
          return pd.DataFrame(results)
```

Initial the model baseline

At this point, we will first use default parameters for the baseline model and will tune the parameters in the later tuning steps if needed.

- **BOGO model**

```
[76]: # implement the model for BOGO offer
drop_cols_prep=['person','offer_id','offer_responded','offer_type']
features,target=data_prep(bogo_offer,drop_cols_prep)
X_train, X_test, y_train, y_test=model_pipeline(features,target)

# initialize the model - baseline is DT model, bogo_1 model is RF model
baseline = DecisionTreeClassifier(criterion='entropy',max_depth=5,random_state=2,min_samples_split=90,min_samples_lea
bogo_1 = RandomForestClassifier(random_state=2,max_depth= 11, max_features= 'auto',min_samples_split= 10,n_estimators

results=run_model(baseline,bogo_1,'bogo_1')
```

```
DecisionTreeClassifier trained on 9829 samples.
MSE_train: 0.1770
MSE_test: 0.1823
Training accuracy:0.8230
Test accuracy:0.8177
             precision    recall  f1-score   support

          0     0.4797    0.2694    0.3450       438
          1     0.8553    0.9366    0.8941      2020

avg / total     0.7884    0.8177    0.7963      2458

RandomForestClassifier trained on 9829 samples.
MSE_train: 0.1670
MSE_test: 0.1786
Training accuracy:0.8330
Test accuracy:0.8214
             precision    recall  f1-score   support

          0     0.4906    0.0594    0.1059       438
          1     0.8287    0.9866    0.9008      2020

avg / total     0.7684    0.8214    0.7591      2458
```

As shown above, the accuracy of both models is good for initial model implementation. But the F1 score is a bit lower than 80% which may be tuned better in the later steps. Although Decision Tree's F1 performs a little better than Random Forest, there's not big hurt to send out some more offers to people who are not going to respond in the end. Therefore, here can still select the random forest with slightly better accuracy right now.

- **Discount Offer model**

```
77]: # instantiate the model for discount offer
drop_cols_prep=['person','offer_id','offer_responded','offer_type']
features,target=data_prep(discount_offer,drop_cols_prep)
X_train, X_test, y_train, y_test=model_pipeline(features,target)

#Initialize the model
discount_1 = RandomForestClassifier(random_state=2,max_depth= 20, max_features= 'auto',min_samples_split= 10,n_estimat
results=pd.concat([results[:],run_model(baseline,discount_1,'discount_1')],axis=1)
```

```
DecisionTreeClassifier trained on 10179 samples.
MSE_train: 0.1371
MSE_test: 0.1277
Training accuracy:0.8629
Test accuracy:0.8723
             precision    recall  f1-score   support

          0     0.0000    0.0000    0.0000       325
          1     0.8723    1.0000    0.9318      2220

avg / total     0.7609    0.8723    0.8128      2545
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWarning: Precision and
F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

```
RandomForestClassifier trained on 10179 samples.
MSE_train: 0.1313
MSE_test: 0.1277
Training accuracy:0.8687
Test accuracy:0.8723
             precision    recall  f1-score   support

          0     0.5000    0.0062    0.0122       325
          1     0.8729    0.9991    0.9317      2220

avg / total     0.8253    0.8723    0.8143      2545
```

As shown above, the random forest performs slightly better than the random forest.

- **Informational offer model**

```
78]: # implement model for informational offer
features,target=data_prep(informational_offer,drop_cols_prep)
X_train, X_test, y_train, y_test=model_pipeline(features,target)

#Initialize the model
info_1 = RandomForestClassifier(random_state=5,criterion='gini',max_depth= 20, max_features= 'auto',min_samples_split=
results=pd.concat([results[:],run_model(baseline,info_1,'info_1')],axis=1)
```

```
DecisionTreeClassifier trained on 5585 samples.
MSE_train: 0.2462
MSE_test: 0.2541
Training accuracy:0.7538
Test accuracy:0.7459
             precision    recall  f1-score   support

          0     0.5000    0.1127    0.1839       355
          1     0.7608    0.9616    0.8495      1042

avg / total     0.6945    0.7459    0.6804      1397

RandomForestClassifier trained on 5585 samples.
MSE_train: 0.2319
MSE_test: 0.2520
Training accuracy:0.7681
Test accuracy:0.7480
             precision    recall  f1-score   support

          0     0.5200    0.1099    0.1814       355
          1     0.7610    0.9655    0.8511      1042

avg / total     0.6997    0.7480    0.6809      1397
```

Model tuning

This section will attempt to tune the parameters of the initial model to get higher performance. In the tuning section, we will first use GridSearch to search for parameters that are likely to get better model performance.

```
[80]: #define Grid Search function
def rand_forest_param_selection(X,y):
    '''
    input:
    - X,y: training datasets for X and y
    output:
    - dictionary with best parameters for random forest model
    '''

    param_grid={'max_features': ['auto', 'sqrt'],
                'max_depth' : [10,15],
                'n_estimators': [10,20,25,30],
                'min_samples_split': [10, 20],
                'min_samples_leaf': [10,15],
                }
    grid_search = GridSearchCV(RandomForestClassifier(random_state=2), param_grid)
    grid_search.fit(X, y)
    grid_search.best_params_
    return grid_search.best_params_
```

```
[81]: #define BOGO dataset
features,target=data_prep(bogo_offer,drop_cols_prep)
X_train, X_test, y_train, y_test=model_pipeline(features,target)

#run Grid Search
rand_forest_param_selection(X_train, y_train)
```

```
[81]: {'max_depth': 10,
 'max_features': 'auto',
 'min_samples_leaf': 10,
 'min_samples_split': 10,
 'n_estimators': 20}
```

Use optimized parameters to rerun the model in the previous steps.

[82]:
```
# use optimized parameters to rerun the model in previous step
# initialize the model
bogo_2 = RandomForestClassifier(random_state=2,max_depth= 10, max_features= 'auto',min_samples_split= 10,n_estimators=2

results=pd.concat([results[:],run_model(baseline,bogo_2,'bogo_2')],axis=1)
```

```
DecisionTreeClassifier trained on 9829 samples.
MSE_train: 0.1770
MSE_test: 0.1823
Training accuracy:0.8230
Test accuracy:0.8177
             precision    recall  f1-score   support

          0     0.4797    0.2694    0.3450       438
          1     0.8553    0.9366    0.8941      2020

avg / total     0.7884    0.8177    0.7963      2458

RandomForestClassifier trained on 9829 samples.
MSE_train: 0.1613
MSE_test: 0.1717
Training accuracy:0.8387
Test accuracy:0.8283
             precision    recall  f1-score   support

          0     0.5870    0.1233    0.2038       438
          1     0.8377    0.9812    0.9038      2020

avg / total     0.7930    0.8283    0.7790      2458
```

Compare the results with the previous initial model.

[83]:
```
results[['RandomForestClassifier_bogo_1','RandomForestClassifier_bogo_2']]
```

[83]:

	RandomForestClassifier_bogo_1	RandomForestClassifier_bogo_2
pred_time	0.029760	0.030382
testing_score	0.821400	0.828316
train_time	0.146926	0.150937
training_score	0.833045	0.838742

[84]:
```
# best model for BOGO offer type
best_model('bogo')
```

```
bogo RF model:
```

[84]:

	pred_time	testing_score	train_time	training_score
RandomForestClassifier_bogo_2	0.030382	0.828316	0.150937	0.838742

As shown above in the comparison, after using tune parameters, the test accuracy slightly improved from 0.833 to 0.838 and the F1 score increased from 0.759 to 0.779.

Do the same steps for discount offer data.

35]:
```
# do the same tuning and refit steps on discount offer
features,target=data_prep(discount_offer,drop_cols_prep)
X_train, X_test, y_train, y_test=model_pipeline(features,target)

# run Grid Search
rand_forest_param_selection(X_train, y_train)
```

35]:
```
{'max_depth': 10,
 'max_features': 'auto',
 'min_samples_leaf': 15,
 'min_samples_split': 10,
 'n_estimators': 30}
```

[86]:
```
# rerun the model with tuned parameters
# initialize the model
discount_2 = RandomForestClassifier(random_state=2,max_depth= 10, max_features= 'auto',min_samples_split= 10,n_esti

results=pd.concat([results[:],run_model(baseline,discount_2,'discount_2')],axis=1)
```

```
DecisionTreeClassifier trained on 10179 samples.
MSE_train: 0.1371
MSE_test: 0.1277
Training accuracy:0.8629
Test accuracy:0.8723
             precision    recall  f1-score   support

          0     0.0000    0.0000    0.0000       325
          1     0.8723    1.0000    0.9318      2220

avg / total     0.7609    0.8723    0.8128      2545
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWarning: Precision ar
F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

```
RandomForestClassifier trained on 10179 samples.
MSE_train: 0.1350
MSE_test: 0.1261
Training accuracy:0.8650
Test accuracy:0.8739
             precision    recall  f1-score   support

          0     1.0000    0.0123    0.0243       325
          1     0.8737    1.0000    0.9326      2220

avg / total     0.8898    0.8739    0.8166      2545
```

ı [87]:
```
results[['RandomForestClassifier_discount_1','RandomForestClassifier_discount_2']]
```

ıt[87]:

	RandomForestClassifier_discount_1	RandomForestClassifier_discount_2
pred_time	0.035388	0.044937
testing_score	0.872299	0.873870
train_time	0.164510	0.231332
training_score	0.868749	0.865016

ı [88]:
```
# best model for discount offer type
best_model('discount')
```

```
discount RF model:
```

ıt[88]:

	pred_time	testing_score	train_time	training_score
RandomForestClassifier_discount_2	0.044937	0.87387	0.231332	0.865016

As shown above in the comparison, after using tune parameters, the test accuracy slightly improved from 0.872 to 0.873 and the F1 score increased from 0.814 to 0.816.

And for the informational offer, do the same step.

[89]:
```
# model tuning for informational offer model
features,target=data_prep(informational_offer,drop_cols_prep)
X_train, X_test, y_train, y_test=model_pipeline(features,target)

#run Grid Search
rand_forest_param_selection(X_train, y_train)
```

[89]:
```
{'max_depth': 10,
 'max_features': 'auto',
 'min_samples_leaf': 10,
 'min_samples_split': 10,
 'n_estimators': 10}
```

[90]:
```
# rerun the model with selected paramenters
info_2 = RandomForestClassifier(random_state=2,max_depth= 10, max_features= 'auto',min_samples_split= 10,n_estimators=
results=pd.concat([results[:],run_model(baseline,info_2,'info_2')],axis=1)
```

```
DecisionTreeClassifier trained on 5585 samples.
MSE_train: 0.2462
MSE_test: 0.2541
Training accuracy:0.7538
Test accuracy:0.7459
             precision    recall  f1-score   support

          0     0.5000    0.1127    0.1839       355
          1     0.7608    0.9616    0.8495      1042

avg / total     0.6945    0.7459    0.6804      1397

RandomForestClassifier trained on 5585 samples.
MSE_train: 0.2378
MSE_test: 0.2470
Training accuracy:0.7622
Test accuracy:0.7530
             precision    recall  f1-score   support

          0     0.5893    0.0930    0.1606       355
          1     0.7599    0.9779    0.8552      1042

avg / total     0.7165    0.7530    0.6787      1397
```

[91]:
```
results[['RandomForestClassifier_info_1','RandomForestClassifier_info_2']]
```

[91]:

	RandomForestClassifier_info_1	RandomForestClassifier_info_2
pred_time	0.019158	0.009385
testing_score	0.748031	0.753042
train_time	0.094609	0.045562
training_score	0.768129	0.762220

[92]:
```
# best model for informational offer type
best_model('info')
```

```
info RF model:
```

[92]:

	pred_time	testing_score	train_time	training_score
RandomForestClassifier_info_2	0.009385	0.753042	0.045562	0.76222

As shown above in the comparison, after using tune parameters, the test accuracy slightly improved from 0.748 to 0.753 and the F1 score increased from 0.681 to 0.678.

View the feature importance

Next, we'll look at the model's result and see if there's any insight into main factors which decide whether customers will respond to offers we could get by investigating feature importance.

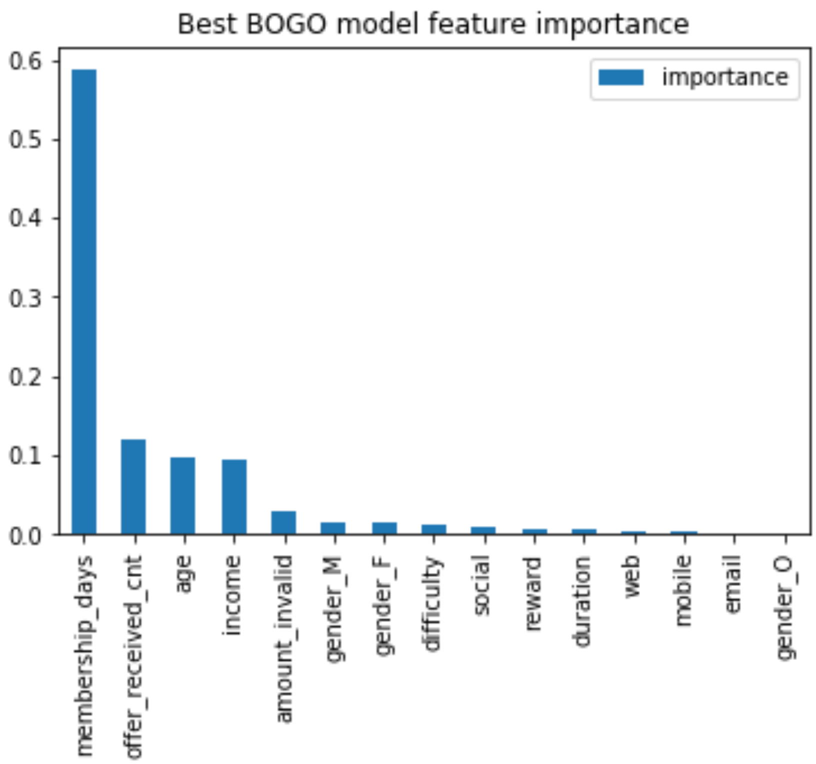
93]:
```
# models summary
best_model('bogo').append([best_model('discount'),best_model('info')]).transpose()
```

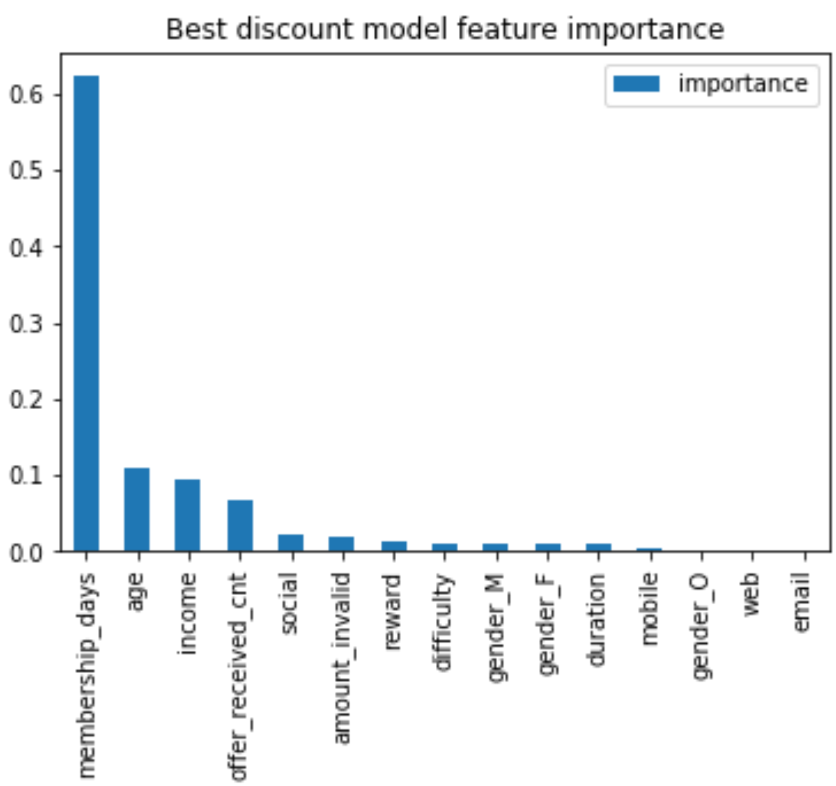
```
bogo RF model:
discount RF model:
info RF model:
```

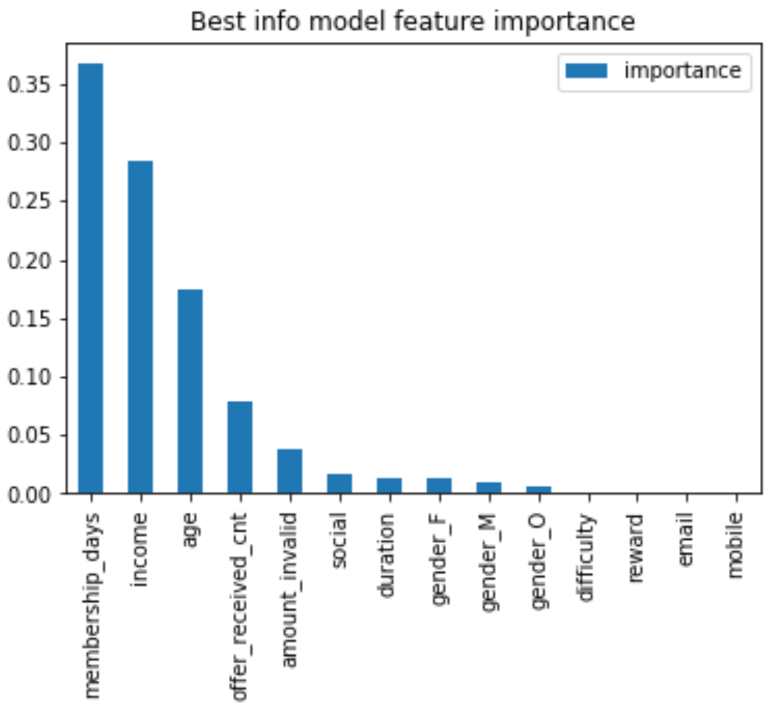
93]:

	RandomForestClassifier_bogo_2	RandomForestClassifier_discount_2	RandomForestClassifier_info_2
pred_time	0.030382	0.044937	0.009385
testing_score	0.828316	0.873870	0.753042
train_time	0.150937	0.231332	0.045562
training_score	0.838742	0.865016	0.762220

Display the feature importance based on the model we have.


Best BOGO model feature importance
importance
0.6
0.5
0.4
0.3
0.2
0.1
0.0
membership_days
offer_received_cnt
age
income
amount_invalid
gender_M
gender_F
difficulty
social
reward
duration
web
mobile
email
gender_O


Best discount model feature importance
importance
0.6
0.5
0.4
0.3
0.2
0.1
0.0
membership_days
age
income
offer_received_cnt
social
amount_invalid
reward
difficulty
gender_M
gender_F
duration
mobile
gender_O
web
email



As shown above, we can see that for all three types of offer, the most important factor that largely affects if the offer will be responded to eventually is the length of membership. That is, the longer the customer as a member of Starbucks, the more likely (s)he will respond to the offer they receive. Then the second and third important factors which affect the possibility of customer's response are age and income which very make sense. Also, the number of offers they received will also affect the response a lot.

Conclusion & Next steps

Conclusion

This project is trying to figure out:

- What factors mainly affect the usage of the offer from the customer? Should the company send out the offer or not?
- How possible will a customer open and use the offer sent to them? Are there any common characteristics of the customers who take the offer?

From the result of the project, it's likely to use machine learning model to predict whether the customer will respond to the offer or not, and the model also shows the main factors such as the length of membership, age, income which highly affect the possibility of customer's responding to the offer.

Next steps

Due to time reasons, I couldn't get a chance to try some other enhancement in the step of model tuning. For example, probably, I can do some more experiment on feature engineering step to see if any other new features can improve the model, also I could also try to reduce some feature to see how it will affect the model performance.

Also, so far the analysis is focused more on customer's who successfully finish the transaction after they received the offer, there should be more insight for the other cases where the customer finishes the transactions regardless of the offer. If we could get any insight into those cases, maybe we can send out more offers to those customers.

In addition, I was thinking if I could do some unsupervised learning on clustering the customers based on information we are given, to see if there are any specific characteristics on a group of customers who will be more likely to respond to the offer.